

Exercise 3

(17/10/2010)



Discussion on Series 2 exercises (already delivered)

- Exercise 2 :

a) x is an odd natural number in the range [15,35]

```
x >= 0 && (x % 2==1) && x <= 35 && x >= 15
```

b) The absolute difference between the a and b is at most 4, where a is less than 15 and b greater than 5

```
abs(a-b)<=4 && a<15 && b>5
```

where abs() is declared in the cmath library

c) b/(b-5) is greater than 1.5 or smaller than -1.5 where b is not 5

```
b!= 5 && (b/(b-5)>1.5 || b/(b-5)<-1.5)
```

or

```
b!= 5 && abs(b/(b-5))>1.5
```

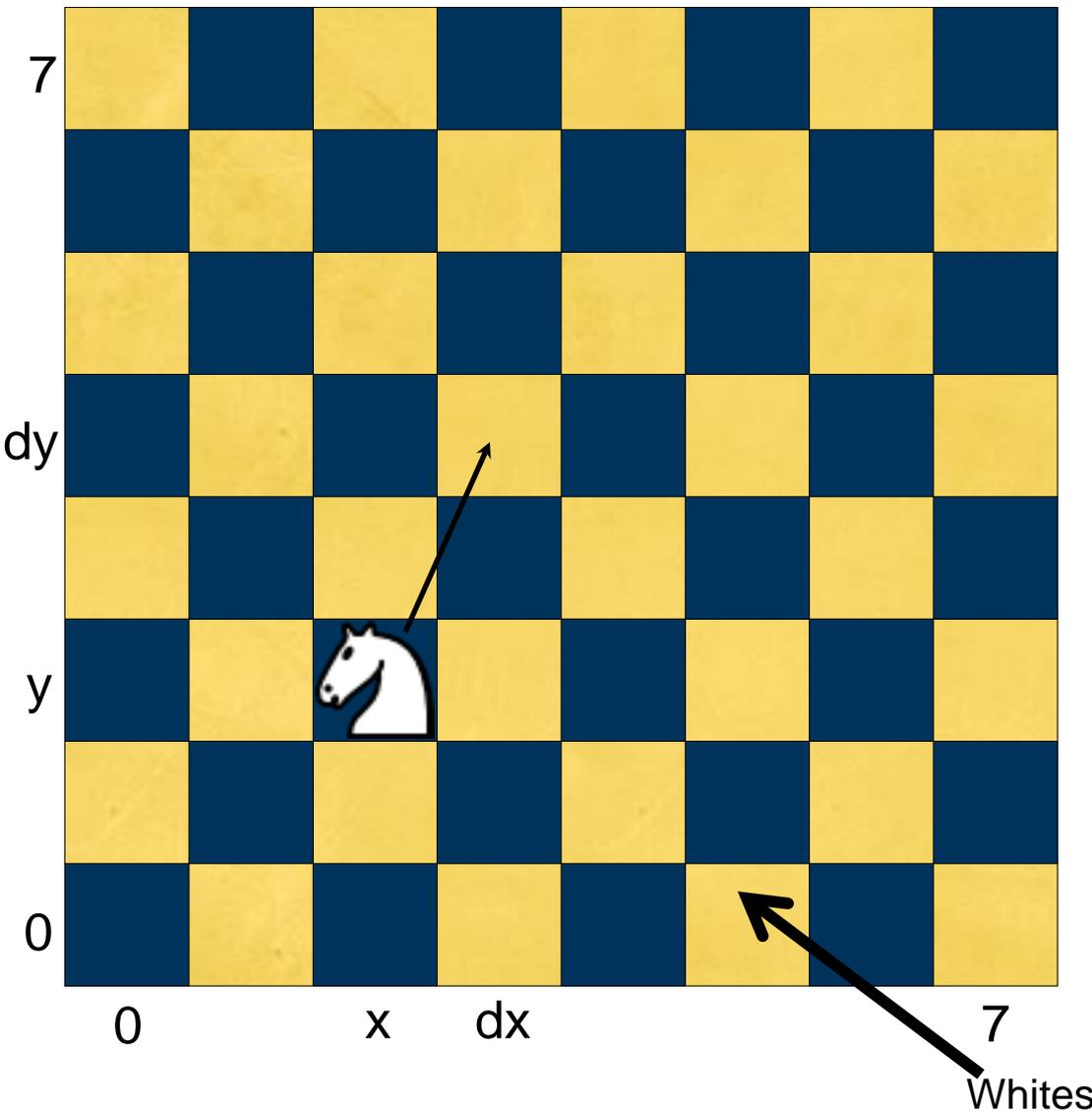
d) Double a equals double b with a tolerance of 10%

```
a>=b*0.9 && a<=b*1.1
```

or

```
abs((a-b)/b) <= 0.1
```

Discussion on Series 2 exercises (already delivered)



```
bool reachable (int piece, int x, int y,
               ... int dx, int dy)
{
  if (x==dx && y==dy) { return false; }
  bool b = (x==dx || y==dy);
  if weisses bauer
    { return x==dx && y==dy-1; }

  if turn { return b; }

  if springer {
    return (abs(y-dy)==2 && abs(x-dx)==1) ||
           (abs(y-dy)==1 && abs(x-dx)==2);
  }

  if dame {
    if (b) {
      return true;
    } else
      laufer

```

Discussion on Series 2 exercises (already delivered)

- Exercise 4 :

```
int main() {  
    // Zahl einlesen  
    double n;  
    double n1;  
    cout << "Bitte geben Sie eine zweistellige  
positive Dezimalzahl ein: ";  
    cin >> n;  
  
    //Zahl pruefen  
    if (n<0 || n>=100) {  
        cout<< "Dies war keine gueltige Zahl"  
<< endl;  
        return 0;  
    }  
}
```

Discussion on Series 2 exercises (already delivered)

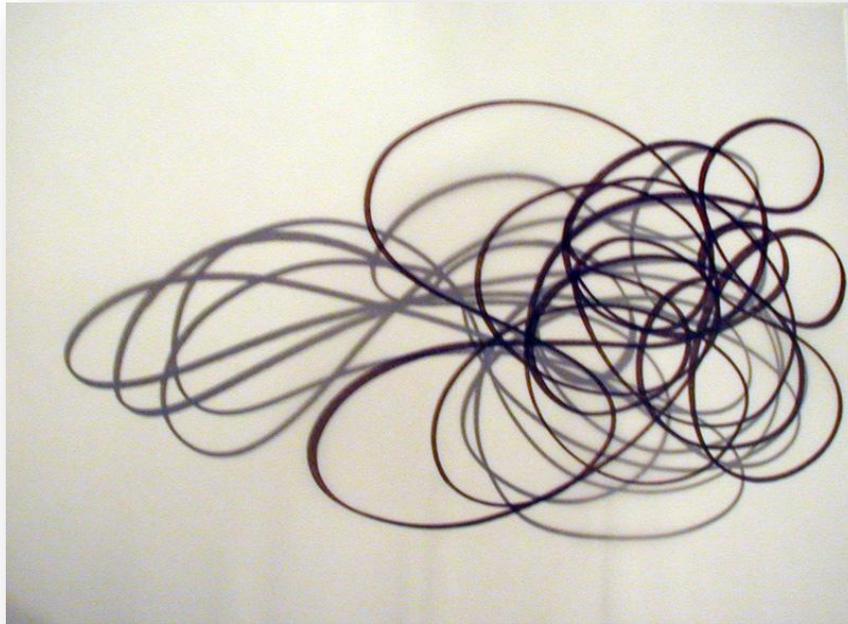
```
//To overcome precision issues
n1 = floor(n*10+0.5);
int ganzzahl = floor(n1/10);

//Ausnahmefaelle abfangen 1,11,12,16,17
if (ausnahmen(ganzzahl)) {
    //Zahl in einzelne Ziffern trennen
    int e = ganzzahl%10;
    int z = ganzzahl/10;
    //Einer-Ziffer ausgeben
    cout << einer(e);
    //Bindungs-und
    if (z > 1 && e > 0) cout << "und";
    //Zehner-Ziffer ausgeben
    cout << zehner(z);
    cout << "komma";

int k = (int) n1 % 10;
if (ausnahmen(k)) {
    cout << einer(e);
}
}
```

LOOPS (SCHLEIFEN)

- We have already how computer programs take decisions (if statement).
- Now we will see how they perform repetitive actions.
- **Loops** are used to repeatedly execute a set of statements till a given statement is true.
- There are 3 kinds of loops in C++ (for, while and do-while loop).



FOR LOOP

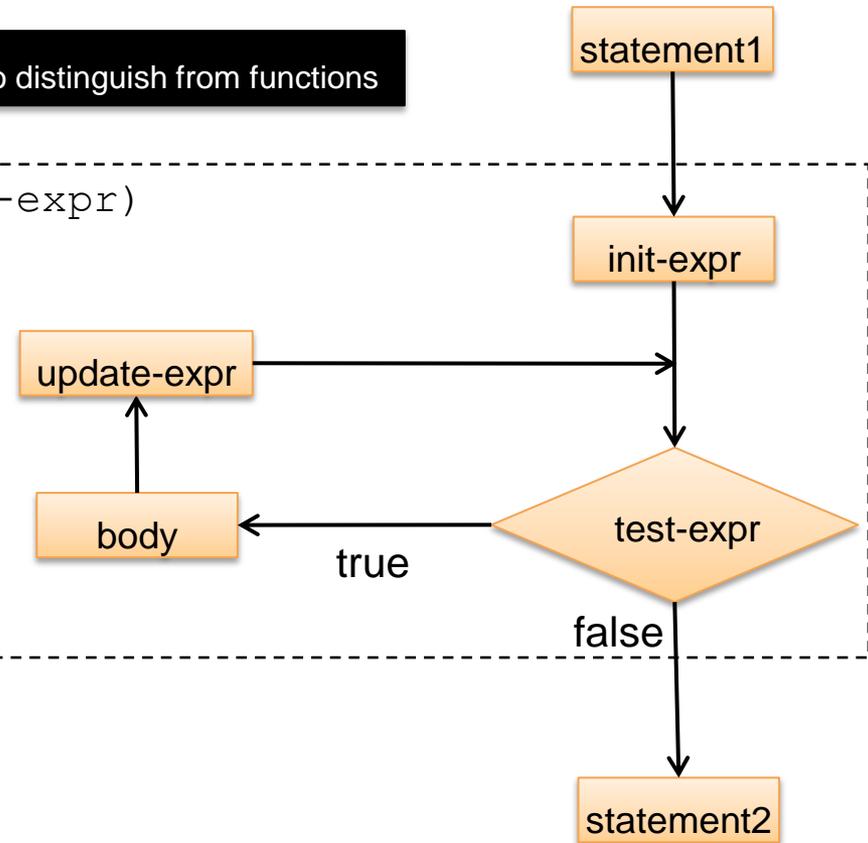
- A `for` loop provides a step-by-step way to perform repeated actions.
- Structure:

Leave a blank here to distinguish from functions

```
statement1;
```

```
for (init-expr; test-expr; update-expr)
```

```
{  
  body;  
  ...  
}
```



```
statement2;
```

With words:

- Set an initial value.
- Perform a test to see whether the loop should continue.
- Execute the statements inside the loop.
- Update values in the test-expression.

FOR LOOP (example)

```
int factorial(int a)
{
```

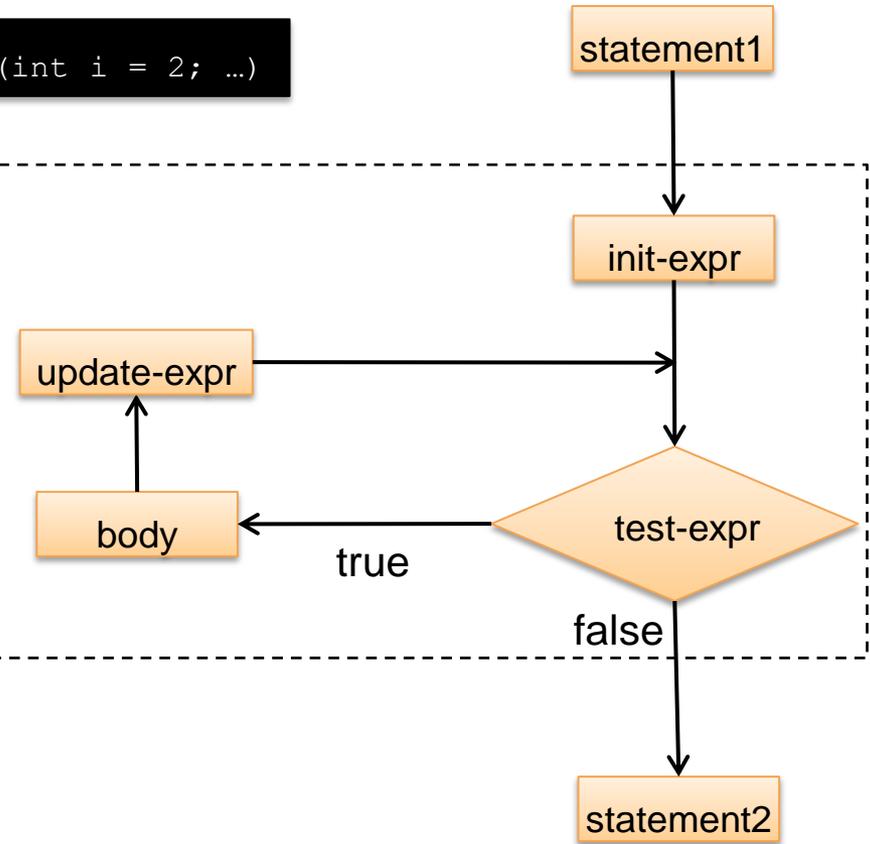
```
    if(a < 0)
        return -1;
    int res = 1;
```

You can also write `for(int i = 2; ...)`

```
    int i;
    for(i = 2; i <= a; i++)
    {
        res *= i;
    }
```

```
    return res;
```

```
}
```



FOR LOOP (Important)

- The **test expression** is evaluated **BEFORE** each loop cycle.
- The loop **never** executes the statements in the body when the test expression is **false**.
- The **update expression** is evaluated at the **END** of the loop, after the body has been executed.
- Typically the update expression increases or decreases the value of the variable in the test expression.
- A for loop can have more than one variable, e.g:

```
for (i=1, j=2; i<=4; i++, j+=2)
{
    cout<<i<<' :'<<j<<endl;
}
```

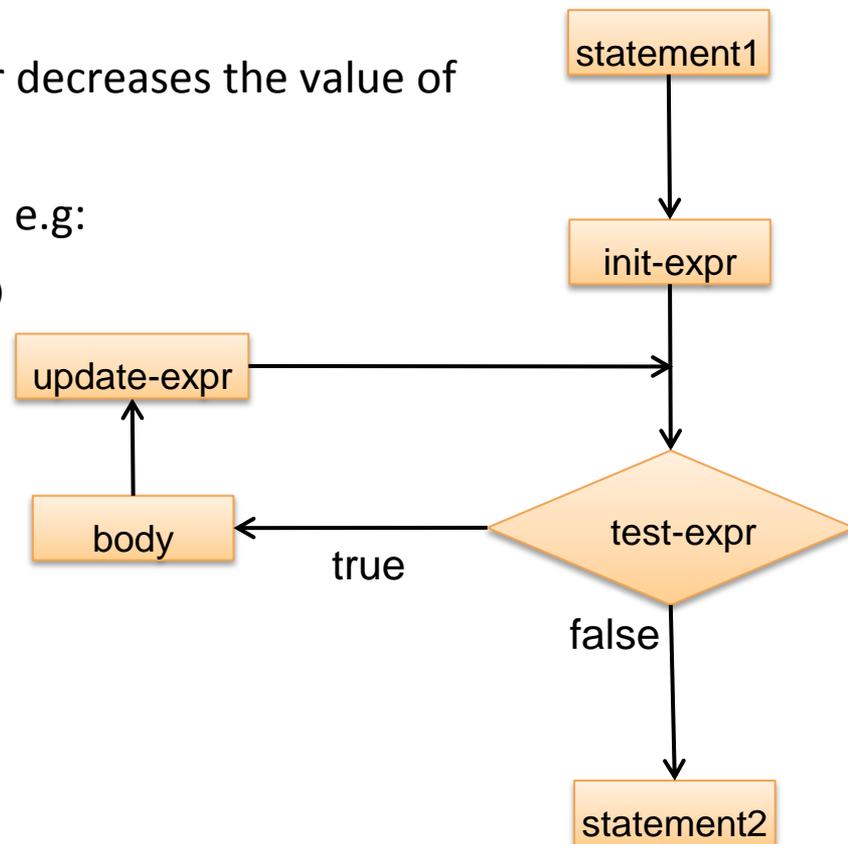
Output:

1:2

2:4

3:6

4:8



WHILE LOOP

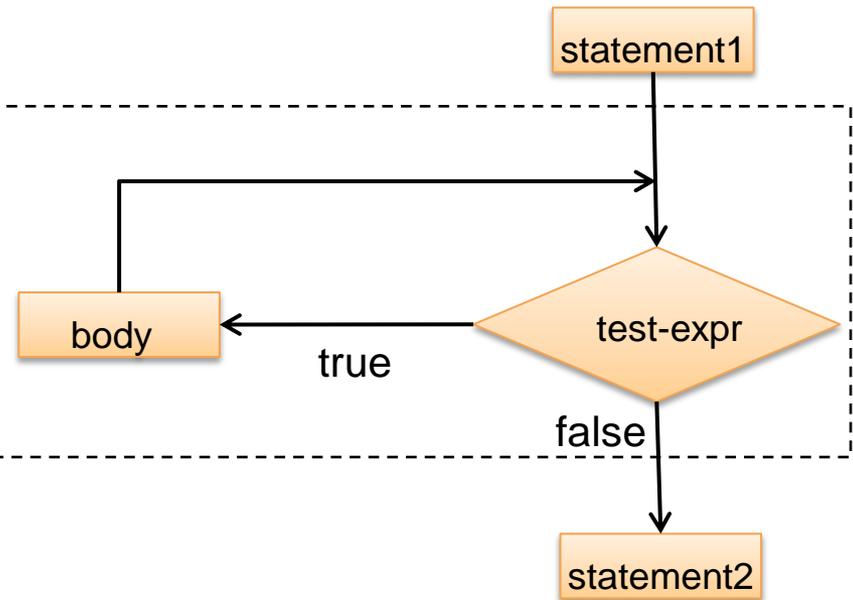
- While loop is a for loop without the initialization and update parts.
- It simply repeats statements **while** the test expression is **true**.

- Structure:

```
statement1;
```

```
while (test-expr)  
{  
    body;  
    ...  
}
```

```
statement2;
```



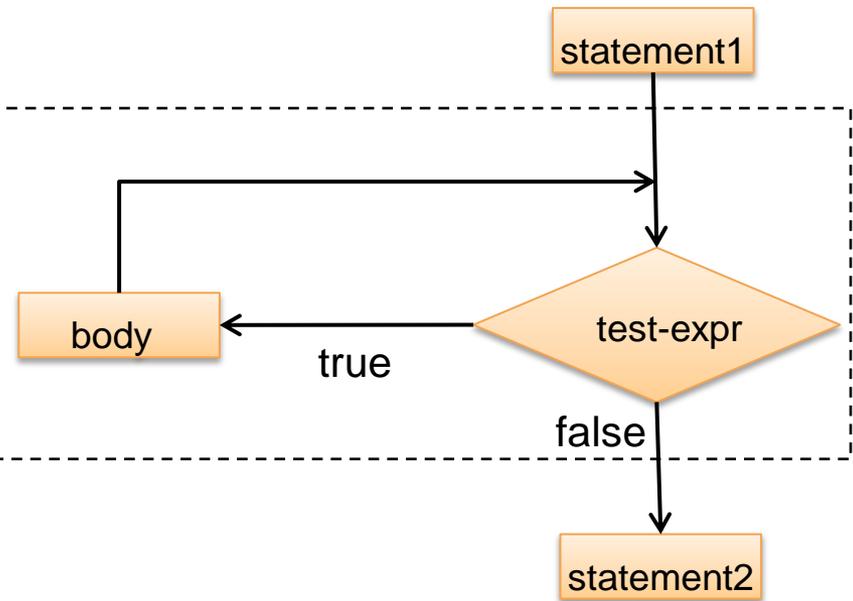
With words:

- While test-expr is true execute statements in the body.

WHILE LOOP (example)

```
include <iostream>
using namespace std;

int main ()
{
    int n;
    cout << "Enter the starting number > ";
    cin >> n;
    while (n>0)
    {
        cout << n << ", ";
        --n;
    }
    cout << "FIRE!\n";
    return 0;
}
```



Output:

Enter the starting number > 8

8, 7, 6, 5, 4, 3, 2, 1, FIRE!

WHILE LOOP (Important)

- **You have to make sure that your while-loop does not run for ever**
- Thus, you have to make sure that in the body, somehow, you have to force the test condition to become false at some point.
- FOR and WHILE loops are essentially equivalent, thus you can use your preferred loop in your code. But:

One nice thing with FOR loops is that it forces you to

- Specify the condition that terminates the loop.
- Initialize that condition before the first test.
- Updates the condition in each loop iteration, before testing again the condition.

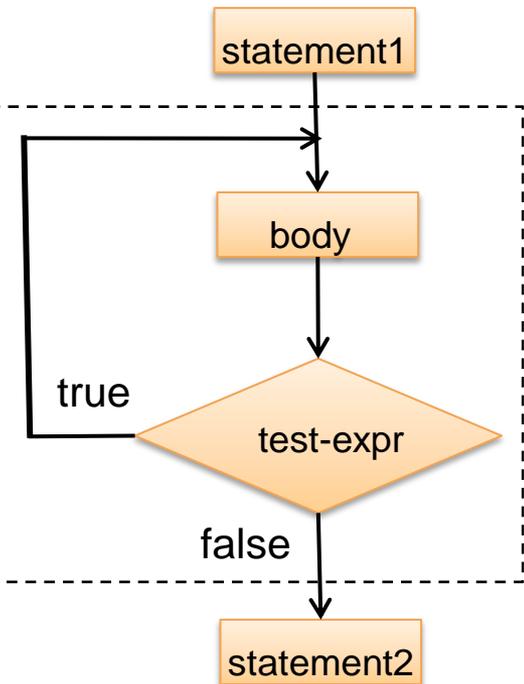
DO-WHILE LOOP

- Exactly the same as a `While` loop, **except that** the test expression is evaluated **AFTER** the execution of the body statement.
- This means that the body will be executed **at least one** time, even if the test expression is never true.

- Structure:

```
statement1;  
do {  
    body;  
    ...  
} while (test-expr)
```

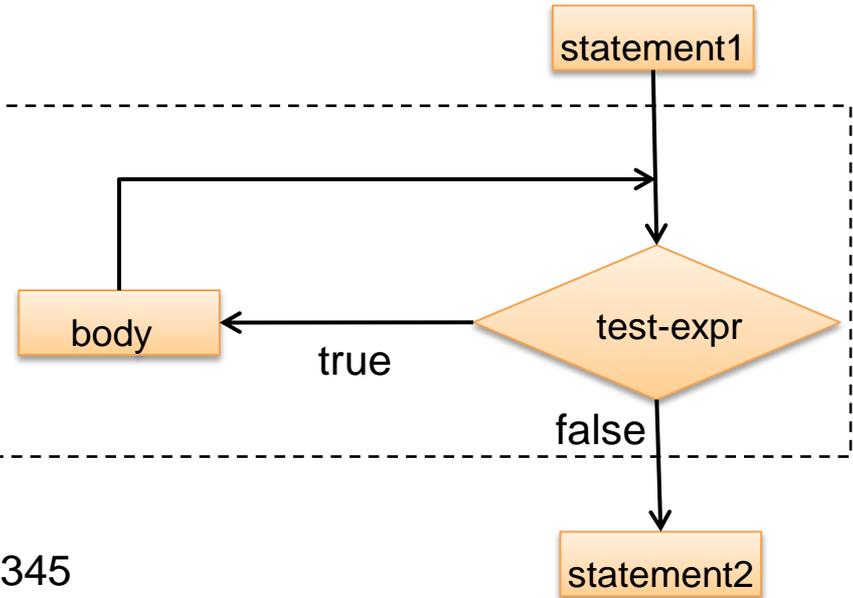
```
statement2;
```



DO-WHILE LOOP (example)

```
#include <iostream>
using namespace std;
```

```
int main ()
{
    unsigned long n;
    do {
        cout << "Enter number (0 to end): ";
        cin >> n;
        cout << "You entered: " << n << "\n";
    } while (n != 0);
    return 0;
}
```



Output:

Enter number (0 to end): 12345

You entered: 12345

Enter number (0 to end): 160277

You entered: 160277

Enter number (0 to end): 0

You entered: 0

LOOP Equivalence

```
int fact1(int a)
{
    if(a < 0)
        return -1;
    int res = 1;
    for(int i = 2; i <= a; i++)
    {
        res *= i;
    }
    return res;
}
```

```
int fact2(int a)
{
    if(a < 0)
        return -1;
    int i = 2, res = 1;
    while(i <= a)
    {
        res *= i;
        i++;
    }
    return res;
}
```

```
int fact3(int a)
{
    if(a < 0)
        return -1;
    int i = 1, res = 1;
    do {
        res *= i;
        i++;
    } while(i <= a);
    return res;
}
```

When you know beforehand how many iterations you have to perform.

When you do not know exactly how many iterations, but you know when you want to stop.

The same as while, if you want to do at least one iteration.

Nested Loops

- You can nest loops one inside the other.
- Used when for one repetition of a process, many repetitions of another process are needed . Some applications are-to print patterns, find sum of a repeating series etc.
- Be careful if within one loop you change the value of a variable that belongs in the test expression of another loop.
- Suppose, for example, that you want to print the following pattern:

1
12
123
1234

```
Code:  
for (i=1; i<=4; i++) {  
    for (j=1; j<=i; j++)  
        cout<<j;  
    cout<<endl;  
}
```

Jump Statements (Forcing exit from a loop)

BREAK

```
// break loop example
#include <iostream>
using namespace std;

int main ()
{
    int n;
    for (n=10; n>0; n--)
    {
        cout << n << ", ";
        if (n==3){
            cout << "countdown abort!";
            break;
        }
    }
    return 0;
}
```

Output: 10, 9, 8, 7, 6, 5, 4, 3, countdown aborted!

CONTINUE

```
// continue loop example
#include <iostream>
using namespace std;

int main ()
{
    for (int n=10; n>0; n--)
    {
        if (n==5)
            continue;
        cout << n << ", ";
    }
    cout << "FIRE!\n";
    return 0;
}
```

Output: 10, 9, 8, 7, 6, 4, 3, 2, 1, FIRE!

Exercise 1 – Differences and Similarities of Loops

- **(1.1)** You are given 3 pieces of code. Find which one prints the letters 'a' to 'f' one after the other on the screen. For the incorrect loops change the code in order to perform this task correctly.
- **(1.2)** Check if the two given pieces of codes are equivalent, i.e. if they print the same on the screen.

```
int i = 0;
while (i<5) {
    i = i+1;
    cout << i*2 << endl;
}
```

```
for (int i=0; i<5; i=i+1) {
    cout << i*2 << endl;
}
```

Exercise 2 – Program analysis

```
bool foo(unsigned int n)
{
    int i = 2;
    bool ip = true;

    while (true){
        if ( i > n/2 )
            break;

        if ( n%i == 0 ){
            ip = false;
            break;
        }
        i++;
    }
    return ip;
}
```

You are given function foo and you are asked to:

- What is the return value of foo(47) ?
- What does foo() calculates?
- Rewrite the function using one for-loop instead of the while loop.

Exercise 3 – Nested Loops

- Write a program (in C++) that prints a Christmas tree on the screen

- Input: The height h of the tree

- Output: (example for $h=5$)



```
  I
 *I*
**I**
***I***
****I****
```

- In order to do that:

- Use nested for-loops, e.g.

```
for-loop for looping over the rows
{
  for-loop for printing empty spaces ' ';
  for-loop for printing the ' I 's;
  for-loop for printing the stars ' * ' ;
}
```

Exercise 4 – Calculate the Euler number

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + \frac{1}{1} + \frac{1}{2!} + \frac{1}{3!} + \dots \approx 2.718281$$

- You have to write a small program that calculates the Euler number using the formula above.
- Store the Euler number as **double** and compute it with a given precision of 10 decimal points.
- Think if you know how many iterations you have to make.
- Think about a proper stopping criterion. Convergence ?

```
step 1: 2
step 2: 2.5
step 3: 2.6666666666666665
step 4: 2.7083333333333333
step 5: 2.7166666666666663
step 6: 2.7180555555555554
step 7: 2.7182539682539684
step 8: 2.71827876984127
step 9: 2.7182815255731922
step 10: 2.7182818011463845
step 11: 2.7182818261984929
step 12: 2.7182818282861687
step 13: 2.7182818284467594
step 14: 2.7182818284582302
step 15: 2.7182818284589949
step 16: 2.7182818284590429
step 17: 2.7182818284590455
step 18: 2.7182818284590455
2.7182818284590455
```

Questions?